IMPROVEMENTS TO REPRESENTATIONS OF COMPRESSED VIDEO

This invention relates to a method of processing of digital information such as video information. This digital video information is either compressed for storage and then later transmission, or is compressed and transmitted live with a small latency. Transmission is for example over the internet.

There is a need for highly efficient compression techniques to be developed to enable transmission of video or other data in real time over the internet because of the restrictions in the bandwidth. In addition, the increasing need for high volume of content and rising end-user expectations mean that a market is developing for live compression at high frame rate and image size.

An object of the invention is to provide such compression techniques.

General background

The video to be compressed can be considered as consisting of a number of frames (at least 1), each made up of individual picture elements, or pixels. Each pixel can be represented by three components, usually either RGB (red, green and blue) or YUV (luminance and two chrominance values). These components can be any number of bits each, but eight bits of each is usually considered sufficient.

The human eye is more sensitive to the location of edges in the Y values of pixels than the location of edges in U and V. For this reason, the preferred implementation here uses the YUV representation for pixels.

The image size can vary, with more pixels giving higher resolution and higher quality, but at the cost of higher data rate. Where the source video is in PAL format, the image fields have 288 lines with 25 frames per second. Square pixels give a source image size of 384 x 288 pixels. The preferred implementation has a resolution of 376 x 280 pixels using the central pixels of a 384 x 288 pixel image, in order to remove edge pixels which are prone to noise and which are not normally displayed on a TV set.

The images available to the computer generally contain noise so that the values of the image components fluctuate. These source images are filtered as the first stage of the compression process. The filtering reduces the data rate and improves the image quality of the compressed video.

A further stage analyses the contents of the video frame-by-frame and determines which of a number of possible types pixel should be allocated to. These broadly correspond to pixels in high contrast areas and pixels in low contrast areas.

The pixels are hard to compress individually, but there are high correlations between each pixel and its near neighbours. To aid compression, the image is split into one of a number of different types of components. The simpler parts of the image split into rectangular components, called "super-blocks" in this application, which can be thought of as single entities with their own structure. These blocks can be any size, but in the preferred implementation described below, the super-blocks are all the same size and are 8 x 8 pixel squares. More structurally complex parts of the image where the connection between pixels further apart is less obvious are split up into smaller rectangular components, called "mini-blocks" in application.

It is apparent that if each super-block is compressed separately, the errors resulting from the compression process can combine across edges between super-blocks thus illustrating the block-like nature of the compression by highlighting edges between blocks, which is undesirable. To avoid this problem, the mini-blocks are tokenised with an accurate representation and these are compressed in a loss free way.

Each super-block or mini-block is encoded as containing YUV information of its constituent pixels.

This U and V information is stored at lower spatial resolution than the Y information, in one implementation with only one value of each of U and V for every mini-block. The super-blocks are split into regions. The colour of each one of these regions is represented by one UV pair.

Real time filtering

The aim is to remove noise from the input video, as noise is by definition hard to compress. The filtering mechanism takes frames one at a time. It compares the current frame with the previous filtered frame on a pixel-by-pixel basis. The value for the previous pixel is used unless there is a significant difference. This can occur in a variety of ways. In one, the value of the pixel in the latest frame is a long way from the value in the previous filtered frame. In another, the difference is smaller, but consistently in the same direction. In another, the difference is even smaller, but cumulatively, over a period of time, has tended to be in the same direction. In these the first two cases, the pixel value is updated to the new value. In the third case, the filtered pixel value is updated by a small amount in the direction of the captured video. The allowable error near a spatial edge is increased depending on the local contrast to cut out the effects of spatial jitter on the input video.

Real time motion estimation

The video frames are filtered into "Noah regions". Thus the pixels near to edges are all labelled. In a typical scene, only between 2% and 20% of the pixels in the image turn out to have the edge labelling. There are three types of motion estimation used. In the first, whole frame pan detection using integer number of pixels is implemented. These motions can be implemented efficiently over the whole image on playback as pixels can be copied to new locations and no blurring is needed. This uses the edge areas from the Noah regions only as the edges contain the information needed for an accurate motion search. The second is sub-pixel motion removal over the whole image. This uses the edge areas from the Noah regions only as the edges contain the information needed for an accurate motion search. The edge pixels in the image, estimated by example from the Noah filtering stage, are matched with copies of themselves with translations of up to e.g., 2 pixels, but accurate to e.g., 1/64 pixel (using a blurring function to smooth the error function) and small rotations. The best match is calculated by a directed search starting at a large scale and increasing the resolution until the required sub-pixel accuracy is attained. This transformation is then applied in reverse to the new image frame and filtering continues as before. These changes are typically ignored on playback. The effect is to remove artefacts caused by camera shake, significantly reducing datarate and giving an increase in image quality. The third for examines local areas of the image. Where a significant proportion of the pixels are updated, for example on an 8x8 pixel block, either motion vectors are tested in this area with patches for the now smaller temporal deltas, or a simplified super-block representation is used giving either 1 or 2 YUVs per block, and patches are made to this.

Real time fade representation

The encoding is principally achieved by representing the differences between consecutive compressed frames. In some cases, the changes in brightness are spatially correlated. In this case, the image is split into blocks or regions, and codewords are used to specify a change over the entire region, with differences with these new values rather than differences to the previous frame itself being used.

Segment Noah regions – find edges

A typical image consists of areas with low contrast and areas of high contrast, or edges. The segmentation stage described here analyses the image and decides whether any pixel is near an edge or not. It does this by looking at the variance in a small area containing the pixel. For speed, in the current implementation, this involves looking at a 3x3 square of pixels with the current pixel at the centre, although future implementations on faster machines can be expected to look at a larger area. The pixels which are not near edges are compressed using an efficient but simple representation which includes multiple pixels – for example 2x2 blocks or 8x8 blocks, which are interpolated on playback. The remaining pixels near edges are represented as either e.g., 8x8 blocks with a number of YUV areas (typically 2 or 3) if the edge is simply the boundary between two or more large regions which just happen to meet here, or as 2x2 blocks with 1Y and one UV per block in the case that the above simple model does not apply e.g., when there is too much detail in the area because the objects in this area are too small.

Miniblockify

The image is made up of regions, which are created from the Noah regions. The relatively smooth areas are represented by spatially relatively sparse YUV values, with the more detailed regions such as the Noah edges being represented by 2x2 blocks which are either uniform YUV, or consist of a UV for the block and maximum Y and a minimum Y, with a codeword to specify which of the pixels in the block should be the maximum Y value and which should be the minimum. To further reduce the datarate, the Y pairs in the non-uniform blocks are restricted to a subset of all possible Y pairs which is more sparse when the Y values are far apart.

Transitions with variable lengths codewords

Compressing video consists in part of predicting what the next frame will as accurately as possible from the available data, or context. Then the (small) unpredictable element is what is sent in the bitstream, and this is combined with the prediction to give the result. The transition methods described here are designed to facilitate this process. On compression, the available context and codeword to compress are passed to the system. This then adds this information to its current distribution (which it is found performs well when it starts with no prejudice as the likely relationship between the context and the output codeword). The distribution output data for this context is calculated and variable length codewords assigned to the outcomes which have arisen. These variable length codewords are not calculated each time the system is queried as the cost/reward ratio makes it unviable, particularly as the codewords have to be recalculated on the player at the corresponding times they are calculated on the compressor. Instead, the codewords are recalculated from time to time. For example, every new frame, or every time the number of codewords has doubled. Recalculation every time an output word is entered for the first time is too costly in many cases, but this is aided by not using all the codeword space every time the codewords are recalculated. Codeword space at the long end is left available, and when new codewords are needed then next one is taken. As these codewords have never occurred up to this point, they are assumed to be rare, and so giving them long codewords is not a significant hindrance. When the codeword space is all used up, the codewords are recalculated. The minimum datarate for Huffman codewords is a very flat and wide minimum, so using the distribution from the codewords which have occurred so far is a good approximation to the optimal. Recalculating the codewords has to happen quickly in a real time system. The codewords are kept sorted in order of frequency, with the most frequent codewords first. The sorting is a mixture of bin sort using linked lists which is O(n) for the rare codewords which change order quite a lot, and bubble sort for the common codewords which by their nature to not change order by very much each time a new codeword is added. The codewords are calculated by keeping a record of the unused codeword space, and the proportion of the total remaining codewords the next data to encode takes. The shorted codeword when the new codeword does not exceed its correct proportion of the available codeword space is used. There is are further constraints – in order to keep the codes as prefix codes and to allow spare space for new codewords, codewords never get shorter in length, and each codeword takes up an integer power of two of the total codeword space. This method creates the new codewords into a lookup table for quick encoding in O(n) where n is the number of sorted codewords.

Memory management

To facilitate Java playback, all the memory used is allocated in one block at the start. As garbage collection algorithms on Java virtual machines are unpredictable, and many stop the system for periods which are long in comparison to the length of a video frame, the invention may use its own memory management system. This involves allocating enough memory for e.g., 2 destination codewords for each source codeword when it is first encountered. New transitions are added as and when they occur, and when the available space for them overflows, the old memory is ignored, and new memory of twice the size is allocated. Although up to half the memory may end up unused, the many rare transitions take almost no memory, and the system scales very well and makes no assumption about the distribution of transitions.

Give compressed codeword for this uncompressed codeword

Every time a codeword occurs in a transition for the second or subsequent time, its frequency is updated and it is re-sorted. When it occurs for the first time in this transition however, it must be defined. As many codewords occur multiple times in different transitions, the destination value is encoded as a variable length codeword each time it is used for the first time, and this variable length codeword is what is sent in the bitstream, preceded by a "new local codeword" header codeword. Similarly, when it occurs for the first time ever, it is encoded raw preceded by a "new global codeword" header codeword. These header codewords themselves are variable length and recalculated regularly, so they start off short as most codewords are new when a new environment is encountered, and they gradually lengthen as the transitions and concepts being encoded have been encountered before.

Video compression (cuts)

Cuts are compressed using spatial context from the same frame.

Cuts, RLE uniform shape, else assume independent and context=CUT_CW.

Cuts -> editable, so needs efficient. First approximation at lower resolution e.g., 8x8.

Cuts – predict difference in mini-block codewords from previous one and uniform flag for current one.

Video compression (deltas)

The deltas can use temporal and spatial context.

Deltas shape – predict shape from uniformness of four neighbours and old shape.

Deltas – predict mini-block codeword differences from uniformness of this mini-block and old mini-block in time.

Datarate reductions

Various simple but effective datarate reduction methods are employed. Noise in the input signal can lead to isolated small changes over the image, whose loss would not be noticed. Isolated changed mini-blocks are generally left out from the bitstream, though if the mini-block is sufficiently different they can still be updated. In addition, small changes in colour in high colour areas are generally ignored as these are almost always caused by noise.

Multi-level gap masks: 4x4, 16x16, 64x64

The bulk of the images are represented mbs and gaps between them. The gaps are spatially and temporally correlated. The spatial correlation is catered for by dividing the image into 4x4 blocks of mbs, representing 64 pixels each, with one bit per mini-block representing whether the mbs has changed on this frame. These 4x4 blocks are grouped into 4x4 blocks of these, with a set bit if any of the mbs it represents have changed. Similarly, these are grouped into 4x4 blocks, representing 128x128 pixels, which a set bit if any of the pixels has changed in the compressed representation. It turns out that trying to predict 16 bits at a time is too ambitious as the system does not have time to learn the correct distributions in a video of typical length. Predicting the masks 4x2 pixels at a time works well. The context for this is the corresponding gap masks from the two previous frames. The transition infrastructure above then gives efficient codewords for the gaps at various scales.

Multiple datarates at once

One of the features of internet or intranet video distribution is that the audience can have a wide range of receiving and decoding equipment. In particular the connection speed may vary widely. In a system such as this designed for transmission across the internet, it helps to support multiple datarates. So the compression filters the image once, then resamples it to the appropriate sizes involving for example cropping so that averaging pixels to make the final image the correct size involves averaging pixels in rectangular blocks of fixed size. There is a sophisticated datarate targeting system which skips frames independently for each output bitstream. The compression is sufficiently fast on a typical modern PC of this time to create modem or midband videos with multiple target datarates. The video is split into files for easy access, and these files may typically be 10 seconds long, and may start with a key frame. The player can detect whether its pre-load is ahead or behind target and load the next chunk at either lower or higher datarate to make use of the available bandwidth. This is particularly important if the serving is from a limited system where multiple simultaneous viewers may wish to access the video at the same time, so the limit to transmission speed is caused by the server rather than the receiver. The small files will cache well on a typical internet setup, reducing server load if viewers are watching the video from the same ISP, office, or even the same computer at different times.

Key frames

The video may be split into a number of files to allow easy access to parts of the video which are not the beginning. In these case, the files may start with a key frame. A key frame contains all information required to start decompressing the bitstream from this point, including a cut-style video frame and information about the status of the transition tables, such as starting with completely blank tables.

Digital Rights Management

DRM is an increasingly important component of a video solution, particularly now content is so readily accessible of the internet. Data typically included in DRM may an expiry data for the video, a restricted set of URLs the video can be played from. Once the compressor itself is sold, the same video may be compressed twice with different DRM data in an attempt to crack the DRM by looking at the difference between the two files. The compression described here is designed to allow small changes to the initial state of the transition or global compression tables to effectively randomise the bitstream. By randomizing a few bits each time a video is compressed, the entire bitstream is randomized each time the video is compressed, making it much harder to detect differences in compressed data caused by changes to the information encoded in DRM.

Miscellaneous

The Y values for each pixel within a single super-block can also be approximated. In many cases, there is only one or part of one object in a super-block. In these cases, a single Y value is often sufficient to approximate the entire super-block's pixel Y values, particularly when the context of neighbouring super-blocks is used to help reconstruct the image on decompression.

In many further cases, there are only two or parts of two objects in a super-block. In these cases, a pair of Y values is often sufficient to approximate the entire super-block's Y values, particularly when the context of the neighbouring super-blocks is used to help reconstruct the image on decompression. In the cases where there are two Y values, a mask is used to show which of the two Y values is to be used for each pixel when reconstructing the original super-block. These masks can be compressed in a variety of ways, depending on their content, as it turns out that the distribution of masks is very skewed. In addition, masks often change by small amounts between frames, allowing the differences between masks on different frames to be compressed efficiently.

Improvements to image quality can be obtained by allowing masks with more than two Y values, although this increases the amount of information needed to specify which Y value to use.

Although the invention has been described with particular reference to video data, it will be appreciated that it could also be applied to other types of data such as audio data.

Brief Descriptions of the Drawings

Figure 1 shows a typical image of 376x280 pixels divided into 8x8 pixel super-blocks.

Figure 2 shows a typical super-block of 8x8 pixels divided into 64 pixels.

Figure 3 shows a typical mini-block of 2x2 pixels divided into 4 pixels.

Figure 4 shows an example image containing two Noah regions and a Noah edge.

Figure 5 shows an example of global accessible context for transition tables.

Figure 6 shows an example of transition tables with local context (LC1 etc) and corresponding resulting values which have been predicted so far.

Figure 7 shows the typical context information for cuts.

Figure 8 shows the typical context information for delta frames.

Figure 9 is a flowchart showing how variable length codewords are generated from a list of codewords sorted by frequency.

Specific description

Video frames of typically 384x288, 376x280, 320x240, 192x144, 160x120 or 128x96 pixels (see figure 1) are divided into pixel blocks, typically 8x8 pixels in size (see figure 2), and also into pixel blocks, typically 2x2 pixels in size, called mini-blocks (see figure 3). In addition, the video frames are divided into Noah regions (see figure 4), indicating how complex an area of the image is.

In one implementation, each super-block is divided into regions, each region in each super-block approximating the corresponding pixels in the original image and containing the following information:

1 Y values (typically 8 bits)

1 U value (typically 8 bits)

1 V value (typically 8 bits)

64 bits of mask specifying which YUV value to use when reconstructing this super-block.

In this implementation, each mini-block contains the following information:

2 Y values (typically 8 bits each)

1 U value (typically 8 bits)

1 V value (typically 8 bits)

4 bits of mask specifying which Y value to use when reconstructing this mini-block.

Temporal gaps

If more latency is acceptable, temporal gaps rather than spatial gaps turn out to be an efficient representation. This involves coding each changed mini-block with a codeword indicating the next time (if any) in which it changes.

Interpolation between Uniform Super-Blocks

Where uniform super-blocks neighbour each other, bilinear interpolation between the Y, and and V values used to represent each block is used to find the Y, U and V values to use for each pixel on playback.

A preferred embodiment of the invention there provides a method of processing digital video information for transmission or storage after compression said method comprising:

reading digital data representing individual picture elements (pixels) of a video frame as a series of binary coded words;

segmenting the image into regions of locally relatively similar pixels and locally relatively distinct pixels;

having a mechanism for learning how contextual information relates to codewords requiring compression and encoding such codewords in a way which is efficient both computationally and in terms of compression rate of the encoded codewords and which dynamically varies to adjust as the relationship between the context and the codewords requiring compression changes and which is computationally efficient to decompress;

establishing a reduced number of possible luminance values for each block of pixels (typically no more than four);

encoding to derive from the words representing individual pixels further words describing blocks or groups of pixels each described as a single derived word which at least includes a representation of the luminance of a block component of at least eight by eight individual pixels (super-block);

establishing a reduced number of possible luminance values for each block of pixels (typically no more than four);

encoding to derive from the words representing individual pixels further words describing blocks or groups of pixels each described as a single derived word which at least includes a representation of the luminance of a block component of typically two by two individual pixels (mini-block);

establishing a reduced number of possible luminance values for each block of pixels (typically one or two);

providing a series of changeable stored masks as a means for indicating which of the possible luminance values are to be used in determining the appropriate luminance value of each pixel for display;

comparing and evaluating the words representing corresponding portions of one frame with another frame or frames in a predetermined sequential order of the elements making up the groups to detect differences and hence changes;

identifying any of the masks which require updating to reflect such differences and choosing a fresh mask as the most appropriate to represent such differences and storing the fresh mask or masks for transmission or storage;

using context which will be available at the time of decompression to encode the masks, the changes in Y values, U values, and V values, and the spatial or temporal gaps between changed blocks, combined with the efficient encoding scheme, to give an efficient compressed real time representation of the video;

using variable length codewords to represent the result of transitions in a way which is nearly optimal from a compression point of view, and computationally very efficient to calculate.